Converting Suffix Trees into Factor/Suffix Oracles

Irena Rusu

Converting Suffix Trees into Factor/Suffix Oracles

Irena Rusu [1]

[1] L.I.N.A., Université de Nantes, 2 rue de la Houssinière, BP 92208, 44322 Nantes, France

— Combinatoire et Bio-informatique —

LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE

RESEARCH REPORT

N° 05.05

October 2005

UNIVERSITÉ DE NANTES

ECOLE DES MINES DE NANTES

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

LINA, Université de Nantes – 2, rue de la Houssinière – BP 92208 – 44322 NANTES CEDEX 3
Tél. : 02 51 12 58 00 – Fax. : 02 51 12 58 12 – http://www.sciences.univ-nantes.fr/lina/

Irena Rusu

Converting Suffix Trees into Factor/Suffix Oracles

18 p.

RROFS6.tex – Converting Suffix Trees into Factor/Suffix Oracles – 4/10/2005 – 13:45

Converting Suffix Trees into Factor/Suffix Oracles

Irena Rusu

Irena.Rusu@univ-nantes.fr

Abstract

The factor/suffix oracle is an automaton introduced by Allauzen, Crochemore and Raffinot. It is built for a given sequence s on an alphabet Σ, and it *weakly recognizes* all the factors (the suffixes, respectively) of s : that is, it certainly recognizes all the factors (suffixes, respectively) of s, but possibly recognizes words that are not factors (not suffixes, respectively) of s. However, it can still be suitably used to solve pattern matching and compression problems. The main advantage of the factor/suffix oracle with respect to other indexes is its size: it has a minimum number of states and a very small (although not minimum) number of transitions.

In this paper, we show that the factor/suffix oracle can be obtained from another indexing structure, the suffix tree, by a linear algorithm. This improves the quadratic complexity previously known for the same task.

General Terms: Algorithms

Additional Key Words and Phrases: indexing structure, factor recognition, suffix recognition

1 Introduction

The need to efficiently solve pattern matching problems lead to the definition of several indexing structures: given a text, these structures allow to store the text, to have a fast access to it and to quickly execute certain operations on data. Suffix arrays, suffix automata, suffix trees are classical structures which can be implemented in linear time with respect to the text size, but still require a too important (although linear) amount of space. Several techniques for reducing the memory space needed by index implementation were developed [3].

One of these techniques proposes to build indexes which are less precise but more space economical than the classical indexes: such a new index validates a set of words which contains all the factors of the text, but possibly contains words that are not factors. We then say that the index performs a *weak factor recognition*.

The factor and suffix oracles introduced in [1] are automata for weak factor recognition. A simple, space economical and linear on-line algorithm to build oracles is given in [1], together with some applications to pattern matching. Other applications to pattern matching, finding maximal repeats and text compression can be found in [4], [5], [6] and [7]. In [8], a combinatorial characterization of the language recognized by the oracles is provided.

Suffix trees and factor/suffix oracles are certainly related to each other. Making this relation more explicit could be a way to obtain results on oracles using already known results on suffix trees. A first, quadratic, algorithm to transform a suffix tree into a suffix oracle was proposed in [2]. We propose here a simple, linear algorithm.

After recalling the main definitions in Section 2, we present in Sections 3, 4, 5 respectively the construction algorithm, the theoretical results insuring its correctness and the reasons of its linear complexity.

2 Trees and automata

Let $s = s_1 s_2 \dots s_m$ be a sequence of length $|s| = m$ on a finite alphabet Σ. Given integers i, j, $1 \leq i \leq j \leq m$, we denote $s[i \dots j] = s_i s_{i+1} \dots s_j$ and call this word a *factor* of s. A *suffix* of s is a factor of s one of whose occurrences ends in position m. The i-th suffix of s, denoted $Suff_s(i)$, is the suffix $s[i \dots m]$ and has length $m + 1 - i$. A *prefix* of s is a factor of s one of whose occurrences starts in position 1. The i-th prefix of s is the prefix $s[1 \dots i]$. Say that a suffix of s is *maximal* if it is not identical to s and it is not the prefix of another suffix of s.

The suffix tree

The *suffix tree* $ST(s)$ of s is a deterministic automaton whose underlying graph is a rooted tree, whose initial state is the root of the tree and whose terminal states are its leaves. The only word accepted in leaf number i, denoted $leaf(i)$, is $Suff_s(i)$. In the non-compact version of the suffix tree, transitions are labeled with letters in Σ. The compact version is obtained from the non-compact version by discarding all states (excepting the root) which have only one outgoing transition and creating new transitions labeled with *factors* of s to replace the missing paths. To insure that every sequence s does have an associated suffix tree, a new character \$ is added to Σ and $ST(s)$ is built for $s\$$ instead of s. Then the only word accepted in $leaf(i)$ $(1 \leq i \leq m+1)$ of $ST(s)$ is $Suff_{s\$}(i)$. See B_0 in Figure 1 for an example.

An important role will be played in the paper by the branches of the suffix tree. The k-th branch $(1 \leq k \leq m+1)$ of $ST(s)$, denoted β_k, is the directed path from the root to $leaf(k)$, used to accept $Suff_{s\$}(k)$. The directed subpath of β_k defined by the states u and v (with v deeper than u) is denoted $\beta_k[u \dots v]$. Its length $|\beta_k[u \dots v]|$ is the number of its transitions, while $\overline{\beta_k}[u \dots v]$ stands for the word spelled along the path $\beta_k[u \dots v]$. Notice that $|\beta_k[u \dots v]| = |\overline{\beta_k}[u \dots v]|$ in the non-compact suffix tree, but $|\beta_k[u \dots v]| \leq |\overline{\beta_k}[u \dots v]|$ in the compact suffix tree.

The factor/suffix oracle

The *factor/suffix oracle* for s is a deterministic automaton for weak factor recognition. It has $m + 1$ states denoted $0, 1, 2, \dots, m$, exactly m *internal transitions* from i to $i + 1$ labeled s_{i+1} and at most $m - 1$ *external transitions* from i to j $(i+1 < j)$ labeled s_j. Consequently, the oracle is *homogeneous*, that is, all the transitions incoming a given state have the same label. Each state is terminal in the factor oracle, while only the states ending the spelling of a suffix of s (including the empty one) are terminal in the suffix oracle (see $B(s)$ in Figure 1 for a suffix oracle). The factor/suffix oracle was introduced in [1] and can be built using an on-line linear algorithm. The algorithm we give here (also proposed in [1]) is quadratic, but more intuitive. In the algorithm, $Oracle(s)$ denotes indifferently the factor or suffix oracle.

Algorithm Build_Oracle [1]
Input: Sequence s.
Output: $Oracle(s)$.

for i from 0 to m do
  create a new state i;
for i from 0 to $m-1$ do
  build a new transition from i to $i+1$ by s_{i+1};
for i from 0 to $m-1$ do
  let u be a minimum length word whose reading ends in state i;
  for all $\sigma \in \Sigma$, $\sigma \neq s_{i+1}$ do
    if $u\sigma$ is a factor of $s[i-|u|+1 \ldots m]$ then
      let j, $i-|u|+1 \leq j \leq m$, be the end position of the first occurrence of $u\sigma$ in $s[i-|u|+1 \ldots m]$;
      build a transition from i to j by σ.

It is shown in [1] that the minimum length word accepted in state i by the factor oracle is unique for each i, $0 \leq i \leq m$.

3 The algorithm

The algorithm we propose in this section transforms the suffix tree of a sequence in the suffix oracle of the same sequence in linear time. A first algorithm for this task was given in [2], but its complexity is quadratic. For the sake of simplicity, the algorithm considers that $ST(s)$ is the non-compact suffix tree, but there are very few changes to perform when $ST(s)$ is compact.

The branch β_1 of $ST(s)$ is called the *main branch* and will be a constant during the algorithm in the following sense: each of the automata B_i ($i \geq 1$) obtained successively from $B_0 = ST(s)$ will have a main branch spelling the same word $s_1 s_2 \ldots s_m\$$. Its end leaf will be denoted $leaf(L_{i+1})$. At each step of the algorithm (see Figure 1), a branch will be *bent* along the main branch. This is done using the contraction operation defined below. The result is an automaton which has less branches and one more external transition. The final automaton $B(s)$ only has one branch (the main one) and as many external transitions as the number of steps that have been performed.

Given two distinct states u, u' of B_i, the operation of *contraction* of u and u' (in *this* order) consists in (a) replacing each transition incoming to u' by a transition with the same start point and incoming to u, and (b) removing the part of the automaton which is disconnected from the root. The resulting state u is called a *contracted state*.

A suffix number $t > L_{i+1}$ is called a *free suffix number* of the automaton B_i ($i \geq 0$) if β_t exists in B_i. A free suffix number t of B_i is *strong* (*weak*, respectively) if $Suff_s(t)$ is a maximal (not maximal, respectively) suffix of s; or, equivalently, if the parent of $leaf(t)$ in $ST(s)$ (non-compact) has exactly one child (at least two children, respectively). The corresponding branches and leaves will be called *free*, *strong* and *weak* respectively. See B_1 in Figure 1: branches 1 and 2 of B_0 have been bent together by contracting u_1 and u'_1, so that their leaves are now identical to $leaf(L_2)$; the suffix number 6 is (free and) strong, the suffix number 7 is (free and) weak, while the suffix numbers 1 and 2 are not free.

The *branching state* of two branches in B_i is the deepest state (that is, the most distant from the root) which is common to the two branches.

In the algorithm below, the notation (x, γ, u) corresponds to a transition from x to u by γ, with $\gamma \in \Sigma$ as long as we refer to the non-compact suffix tree.

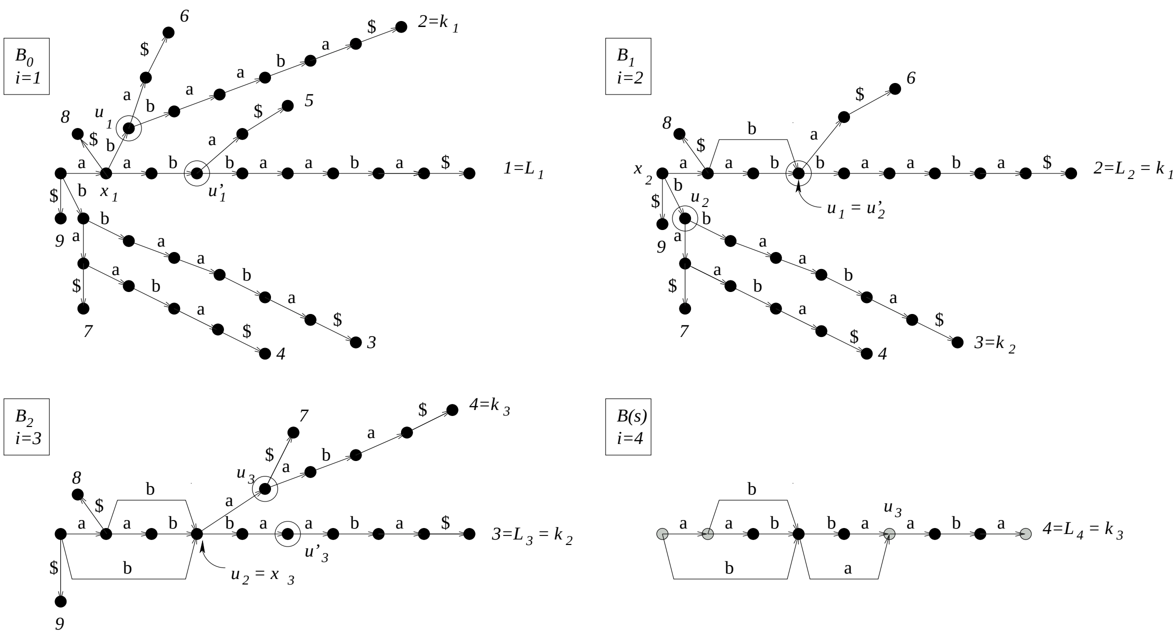

Figure 1: *Execution of the algorithm* **ST-to-SO** *on the suffix tree of* $s = aabbaaba$*. The grey states are terminal. The main branch is the horizontal path labeled* $aabbaaba\$$*.*

The execution of the algorithm **ST-to-SO** is shown in Figure 1.

Algorithm ST-to-SO
Input: $ST(s)$
Output: automaton $B(s)$
$B_0 := ST(s)$;
β_0:=β_1; /*main branch*/
$L_1 := 1$; /*the leaf number of the main branch*/
$i := 1$;
while (B_{i-1} has at least one strong branch) do
  let $k_i > L_i$ be min. s. t. β_{k_i} is strong in B_i ;
  $B_i := bend(B_{i-1}, k_i, L_i)$;
  $L_{i+1} := k_i$;
  $i := i + 1$
mark the parent of each weak leaf as terminal;
remove all the weak leaves;
mark the parent of $leaf(L_i)$ as terminal;
remove $leaf(L_i)$;
Return $B(s)$.

Function bend(B_{i-1}, k_i, L_i**)**
$x_i :=$ the branching state of β_0 and β_{k_i};
$(x_i, \gamma_i, u_i) :=$ the transition going out from x_i on β_{k_i};
$u'_i :=$ the state of β_0 such that $|\overline{\beta_0}[u'_i \ldots leaf(L_i)]|$ and $|\overline{\beta_{k_i}}[u_i \ldots leaf(k_i)]|$ are identical;
contract u_i and u'_i;
$\beta_0 :=$ the new path spelling $s\$$;
Return B_i

Remark 1 If k is a weak suffix number of B_i ($i \geq 0$), then so are $k+1, \ldots, m$. Consequently, every weak suffix number is greater than every strong suffix number. Thus, according to the algorithm **ST-to-SO**, in each B_i one has $L_{i+1} < k_{i+1} < t$, for each leaf number $t \neq L_{i+1}, k_{i+1}$ which exists in B_i.

4 Theoretical results

In this section (Theorem 2) we prove that the automaton $B(s)$ built by the algorithm **ST-to-SO** is identical to the suffi x oracle $SO(s)$ of s. The proofs (and thus the results) are valid in both cases with respect to $ST(s)$ (compact or non-compact suffi x tree).

We will call a *step* of the algorithm **ST-to-SO** any execution of the *while* loop. The i-th step is then executed on B_{i-1} and is identifi ed by the 7-tuple $(i, L_i, k_i, x_i, \gamma_i, u_i, u'_i)$, where L_i is the end state of the main branch at the beginning of the i-th step and the other variables are computed in the function *bend*. The automaton obtained *at the end* of the i-th step is denoted B_i. Thus the main branch β_0 of B_i ends with the state $L_{i+1} = k_i$ and B_i still contains the states x_i, u_i but not u'_i. Moreover, it contains states $k_{i+1}, x_{i+1}, u_{i+1}, u'_{i+1}$. In B_i, we have that: (1) the branches β_j with $1 \leq j \leq L_{i+1} - 1$ do not exist anymore (either they were bent, or they have been removed during a contraction); (2) a branch β_j with $L_{i+1} < j \leq m + 1$ either exists and is free (strong or weak) or does not exist anymore (it has been removed during a contraction). The notation β_k is not ambiguous and will be the same in all the automata B_i built during the algorithm.

There are mainly two affi rmations to justify before proving Theorem2:

i) The contracted states $u_1, u_2, \ldots, u_i$ are placed in this order on the main branch of $B(s)$ (the equality is possible).

ii) During the contraction of u_i and u'_i, some branches of B_{i-1} are removed, but the suffi xes corresponding to these branches are still accepted by B_i.

To prove affi rmation i), it is essential to compare the lengths of the words $\overline{\beta_{k_i}}[x_i \ldots leaf(k_i)]$, for all i. A surprising remark is that for any branch β_{k_i} which is bent during the algorithm, one can identify very early, in $ST(s)$, the state x_i which will be computed by the algorithm. Thus, one can equally compute the length of $\overline{\beta_{k_i}}[x_i \ldots leaf(k_i)]$. This is proved in (A1) of Theorem 1.

Affi rmation ii) is based on the remark that each branch removed during a contraction has a *copy* in the subtree of B_{i-1} rooted in u_i, which allows to substitute the missing branch in B_i. This is proved in (A4) of Theorem 1.

Affi rmations (A2), (A3) and (A5) in Theorem1 are technical requirements.

To give Theorem 1, we need several defi nitions. Let v be a state and t be a free suffi x number of B_i. Then we denote:

- $min(v, B_i)$ the minimum length word read along a path joining the root to v in B_i.
- $lm(v, B_i)$ the length of $min(v, B_i)$.
- $h(t, B_i)$ the deepest branching state of β_t with some branch β_j such that $0 \leq j < t$, in B_i.
- $minb(t, B_i) = |\beta_t[h(t, B_i) \ldots leaf(t)]|$.
- $minw(t, B_i) = |\overline{\beta_t}[h(t, B_i) \ldots leaf(t)]|$.

In Figure 1, $h(3, B_i)$ is the root for $i = 1, 2$, and $minb(3, B_i) = 7 = minw(3, B_i)$ for $i = 1, 2$ (but $minb(3, B_i) = 2$ in the compact version of $ST(s)$).

For the theorem below, see Figure 2.

Theorem 1 *The automata B_i ($i \geq 0$) obtained in Algorithm* **ST-to-SO** *have the following properties:*

(A1) Contraction progress

a) The values $h(t, B_i)$, $minb(t, B_i)$, $minw(t, B_i)$ are constant over all i such that t is a free suffix of B_i (denote them $h(t)$, $minb(t)$, $minw(t)$ respectively). Moreover, if $1 < t < t'$ then $minw(t) \geq minw(t')$. In addition, in the i-th step ($i \geq 1$) of the Algorithm **ST-to-SO**, *$x_i = h(k_i)$.*

b) In B_i ($i \geq 2$), u_i is situated on the path $\beta_0[u_{i-1} \ldots leaf(L_{i+1})]$. The case where u_i replaces u_{i-1} is possible and holds if and only if $minw(k_i) = minw(k_{i-1})$.

c) $\overline{\beta_0}[u'_i \ldots leaf(L_i)]$ and $\overline{\beta_{k_i}}[u_i \ldots leaf(k_i)]$ are identical words in the definition of the function bend, so that β_0 is well defined in this function.

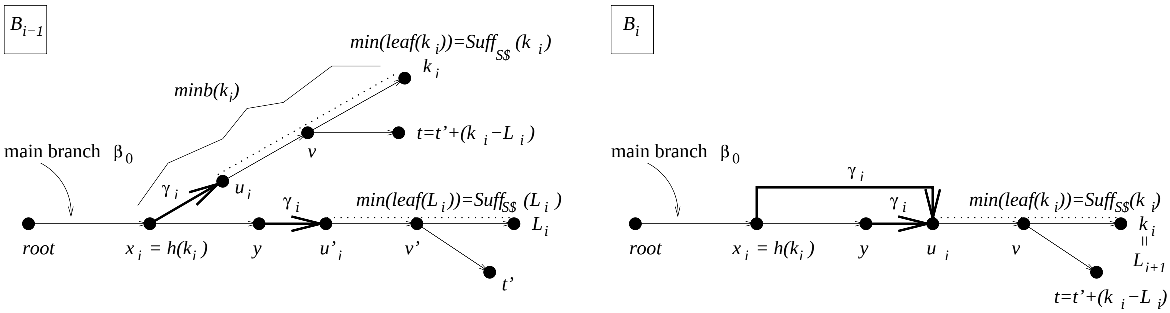

Figure 2: *Step i in Algorithm* **St-to-SO**. *The dashed lines represent paths spelling the same word. The thick arrows are arcs, while the thin arrows are paths.*

(A2) Minimum words

For each state v in B_i, $min(v, B_i)$ is unique and is the same in all the automata B_i containing v (denote $min(v)$ this word and $lm(v)$ its length). Moreover, $lm(u_i) < lm(u'_i)$ in each step i of the algorithm. Furthermore, if x is the deepest contracted state which precedes v or is equal to v (assuming such a state exists), then $min(x)$ is a prefix of $min(v)$. In addition, for each suffix number t of B_i, $min(leaf(t)) = Suff_{s\$}(t)$.

(A3) General properties

The automaton B_i ($i \geq 0$) is deterministic and homogeneous.

(A4) Branch substitution

Let t' be a strong (weak, respectively) suffix number of B_{i-1}. Denote v' the branching state of $\beta_{t'}$ and β_0 in B_{i-1} and assume that v' is a state of the path $\beta_0[u'_i \dots leaf(L_i)]$. Then:

a) the branch $t = t' + (k_i - L_i)$ is a free (weak, respectively) branch of B_{i-1} whose branching state v with β_{k_i} has the properties: $\overline{\beta_{k_i}}[u_i \dots v] = \overline{\beta_0}[u'_i \dots v']$ and $\overline{\beta_t}[v \dots leaf(t)] = \overline{\beta_{t'}}[v' \dots leaf(t')]$.

b) $Suff_{s\$}(t')$ is still accepted by B_i, in $leaf(t)$.

(A5) Main terminal state

For each suffix number $k \in \{1, 2, \dots, m+1\}$, $Suff_{s\$}(k)$ is accepted in a leaf of B_i.

Proof. The affirmations are proved in the given order.

Proof of (A1). To prove (A1a), we use induction on i. Denote $z(t, B_i)$ the minimum number j, $0 \leq j < t$, such that $h(t, B_i)$ is, in B_i, a branching state of β_t and of β_j. Let $root$ be the root of B_i (it is the same state for all the values i).

When $i = 0$, $B_0 = ST(s)$ and only have to show that $minw(t, B_0) \geq minw(t+1, B_0)$ for all t, $1 < t \leq m$. To see this, let $v = h(t, B_0)$ be the the branching state of $\beta_{z(t,B_0)}$ and β_t. Then the $lm(v, B_0)$ first characters of $Suff_{s\$}(t)$ and $Suff_{s\$}(z(t, B_0))$ are common, while the $(lm(v, B_0) + 1)$-th character is different. Consequently, $Suff_{s\$}(t+1)$ and $Suff_{s\$}(z(t, B_0) + 1)$ have their first $lm(v, B_0) - 1$ characters in common, while the $lm(v, B_0)$-th character is different. Call v' the common state of β_{t+1} and $\beta_{z(t,B_0)+1}$ where the reading of their common prefix of length $lm(v, B_0) - 1$ ends. Then, since we are in $B_0 = ST(s)$ and there is a unique path joining the root to v', we have that $lm(v', B_0) = lm(v, B_0) - 1$. Moreover, since $z(t, B_0) + 1 < t + 1$, $h(t+1, B_0)$ is either v' or is deeper than v' on β_{t+1}. Consequently:

$$\begin{aligned} minw(t+1, B_0) &\leq |\overline{\beta_{t+1}}[v' \dots leaf(t+1)]| = m + 2 - (t+1) - lm(v', B_0) = \\ &= m + 2 - t - lm(v, B_0) = |\overline{\beta_t}[v \dots leaf(t)]| = minw(t, B_0). \end{aligned}$$

Affirmation (A1a) is proved in the case $i = 0$.

We continue the induction with the general step. To show (A1a) for B_i, let $t \neq k_i, L_i$ be a free suffix number of B_{i-1}, and call v the state closest to $leaf(t)$ among the branching states of β_t with β_{k_i} and with β_{L_i} respectively. Note that by Remark 1, $t > k_i > L_i$.

$i)$ If v is situated on $\beta_{k_i}[u_i \ldots leaf(k_i)]$, then either $z(t, B_{i-1}) = k_i$, or $k_i < z(t, B_{i-1}) < t$ and in this case the branching state $h(t, B_i)$ of $\beta_{z(t,B_{i-1})}$ and β_t is an internal state of $\beta_t[v \ldots leaf(t)]$. After the contraction, $h(t, B_i) = h(t, B_{i-1})$ (since the subtree rooted at u_i is not modified by the contraction). Moreover, the definitions of $minb(.,.)$ and $minw(.,.)$ insure that these values are not changed neither.

$ii)$ If v is situated on $\beta_0[x_i \ldots u'_i]$, then either $z(t, B_{i-1}) = L_i$ or $k_i < z(t, B_{i-1}) < t$ and in this case the branching state $h(t, B_i)$ of $\beta_{z(t,B_{i-1})}$ and β_t is an internal state of $\beta_t[v \ldots leaf(t)]$. After the contraction, none of the values $h(.,.)$, $minb(.,.)$, $minw(.,.)$ changes.

$iii)$ If v is situated on $\beta_0[root \ldots x_i]$, then nothing changes after the contraction.

$iv)$ If v is situated on $\beta_0[u'_i \ldots leaf(L_i)]$, then after the contraction t is no more a free suffix number.

Thus, the values $h(t, B_i)$, $minb(t, B_i)$, $minw(t, B_i)$ are constant over all the values i such that t is a free suffix of B_i. With the new notation $minw(t, B_i) = minw(t)$, we have that $minw(t) \geq minw(t')$ holds when $t < t'$ (since we proved it in B_0). Moreover, during the i-th step, $z(k_i, B_{i-1})$ must be 0, obtained during a contraction according to case $i)$ above. Then $h(k_i)$, which is by definition the branching state of $\beta_{z(k_i,B_{i-1})}$ and β_{k_i}, is equal to x_i.

To show (A1b), notice that after the contraction yielding B_{i-1} (involving branches L_{i-1} and k_{i-1}), the last state of the main branch is $leaf(L_i) = leaf(k_{i-1})$. Moreover, we have that $k_{i-1} > L_{i-1}$ and, similarly, that $k_i > L_i = k_{i-1}$. Thus, by (A1a), $minw(k_i) \leq minw(k_{i-1})$. Now, $x_{i-1} = h(k_{i-1})$ and $x_i = h(k_i)$ by (A1a), which imply that $minw(k_{i-1}) = |\overline{\beta_{k_{i-1}}}[x_{i-1} \ldots leaf(k_{i-1})]|$ (in B_{i-2}) and $minw(k_i) = |\overline{\beta_{k_i}}[x_i \ldots leaf(k_i)]|$ (in B_{i-1}).

With the main branch β_0 in B_{i-1}, we have in B_{i-1} :

$$\begin{aligned}
|\overline{\beta_0}[u'_i \ldots leaf(L_i)]| &= |\overline{\beta_{k_i}}[u_i \ldots leaf(k_i)]| = minw(k_i) - 1 \leq minw(k_{i-1}) - 1 = \\
&= |\overline{\beta_{k_{i-1}}}[u_{i-1} \ldots leaf(k_{i-1})]| = |\overline{\beta_0}[u_{i-1} \ldots leaf(L_i)]|
\end{aligned}$$

since $\beta_{k_{i-1}}$ and β_0 are identical from u_{i-1} to the leaf, as insured by the algorithm, and since $L_i = k_{i-1}$ as already mentioned. Then (A1b) is proved.

To show (A1c), notice that by (A1b) there is no contracted state between u'_i and L_i on the main branch of B_{i-1}, so that $\overline{\beta_0}[u'_i \ldots leaf(L_i)]$ and $\overline{\beta_{k_i}}[u_i \ldots leaf(k_i)]$ are suffixes of the same length of $s\$$. They are thus identical and the definition of β_0 in the function *bend* is correct.

Proof of (A2). We use induction on i. When $i = 0$, (A2) is obviously true since for each state v of $ST(s)$ there is exactly one path from the root to v. When $v = leaf(t)$ for a suffix number t, then by the definition of the suffix tree, the unique path from the root to v accepts $Suff_{s\$}(t)$.

Now, let $i \geq 1$ and assume that (A2) is true for B_{i-1}. Consider the i-th step, which transforms B_{i-1} into B_i (see Figure 2). According to the algorithm, the only states v of B_i whose minimum length word could be changed during the contraction are situated in the subtree rooted at u_i. We show that $min(u_i, B_{i-1}) = min(u_i, B_i)$ and this is sufficient to deduce that (A2) is true, since then the states v in the subtree of u_i satisfy $min(v, B_{i-1}) = min(v, B_i)$ and obviously $min(u_i, B_i)$ is a prefix of $min(v, B_i)$.

We show that:

$$lm(u_i, B_{i-1}) < lm(u'_i, B_{i-1}). \tag{1}$$

In order to recognize $min(leaf(k_i))$ ($min(leaf(L_i))$ respectively) one has to use the state u_i (u'_i respectively) so that we have by induction in B_{i-1} and using (A1b) to deduce that there is no contracted state on $\beta_0[u'_i \ldots leaf(L_i)]$:

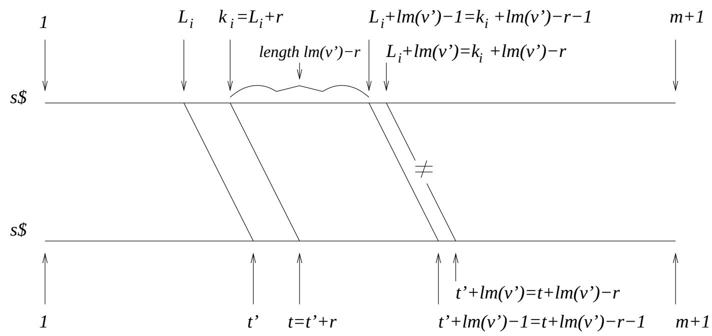

Figure 3: *Identical characters in the proof of (A4a)*

$$lm(u_i, B_{i-1}) = lm(leaf(k_i), B_{i-1}) - |\overline{\beta_{k_i}}[u_i \dots leaf(k_i)]| = m + 2 - k_i - |\overline{\beta_{k_i}}[u_i \dots leaf(k_i)]|$$

and

$$lm(u'_i, B_{i-1}) = lm(leaf(L_i), B_{i-1}) - |\overline{\beta_0}[u'_i \dots leaf(L_i)]| = m + 2 - L_i - |\overline{\beta_0}[u'_i \dots leaf(L_i)]|.$$

According to the algorithm, $|\overline{\beta_{k_i}}[u_i \dots leaf(k_i)]| = |\overline{\beta_0}[u'_i \dots leaf(L_i)]|$. We deduce that $lm(u'_i, B_{i-1})$ $-lm(u_i, B_{i-1}) = k_i - L_i > 0$ and (1) is proved. Therefore, all the words whose reading ends in u_i in B_i and in u'_i in B_{i-1} are longer than $min(u_i, B_{i-1})$, therefore $min(u_i, B_i) = min(u_i, B_{i-1})$.

Proof of (A3). To start the induction (again on the number i of the current step), when $i = 0$ the automaton B_i is $ST(s)$ which is by definition deterministic and homogeneous.

Now, assuming (A3) holds for B_{i-1}, it is easy to note that B_i will be deterministic, since each state in B_i keeps the outgoing transitions it had in B_{i-1}.

To show that B_i is homogeneous, we have to show that in B_{i-1} the transition incoming to u'_i along the main branch is labeled γ_i. This is easy too, since the word read along the main branch is always $s\$$. Since $\overline{\beta_{k_i}}[x_i \dots leaf(k_i)]$ is a suffix of $min(leaf(k_i))$, and thus of s (by (A2)), and since $\overline{\beta_{k_i}}[u_i \dots leaf(k_i)] = \overline{\beta_0}[u'_i \dots leaf(L_i)]$ by (A1c), the characters preceding u_i and u'_i respectively must be the same.

Proof of (A4). According to Remark 1 one has $t' > L_i$ in B_{i-1}. We first prove statement (A4a). Consider the most recently contracted state, u_{i-1}. By (A1b) we deduce that u_{i-1} is also the deepest contracted state on the main branch, and that $u_{i-1} = v'$ if and only if $u_{i-1} = u'_i = v'$. Thus, by (A2) and because $L_i = k_{i-1}$, we have:

$$Suff_{s\$}(L_i) = min(u_{i-1})\overline{\beta_0}[u_{i-1} \dots leaf(L_i)] = min(v')\overline{\beta_0}[v' \dots leaf(L_i)],$$

and

$$Suff_{s\$}(t') = min(u_{i-1})\overline{\beta_{t'}}[u_{i-1} \dots leaf(t')] = min(v')\overline{\beta_{t'}}[v' \dots leaf(t')],$$

so that $Suff_{s\$}(L_i)$ and $Suff_{s\$}(t')$ have their $lm(v')$ first characters in common, but not the $(lm(v') + 1)$-th. Thus (see Figure 3):

$$s_{L_i} = s_{t'}, s_{L_i+1} = s_{t'+1}, \dots, s_{L_i+lm(v')-1} = s_{t'+lm(v')-1}, \tag{2}$$

$$\text{but } s_{L_i+lm(v')} \neq s_{t'+lm(v')}. \tag{3}$$

Furthermore, $|\overline{\beta_{k_i}}[u_i \dots leaf(k_i)]| = |\overline{\beta_0}[u'_i \dots leaf(L_i)]| \geq |\overline{\beta_0}[v' \dots leaf(L_i)]|$, so that $m + 2 - k_i - lm(u_i) = m + 2 - k_i - (lm(x_i) + 1) \geq m + 2 - L_i - lm(v')$. We deduce

$$L_i + lm(v') \geq k_i + lm(x_i) + 1 > k_i, \tag{4}$$

while $k_i > L_i$ by definition. Therefore, k_i can be written like $k_i = L_i + r$, with $r = k_i - L_i < lm(v')$, and formulas (2), (3) imply :

$$s_{k_i} = s_{t'+r}, s_{k_i+1} = s_{t'+r+1}, \ldots, s_{k_i+lm(v')-r-1} = s_{t'+lm(v')-1},$$

$$\text{but } s_{k_i+lm(v')-r} \neq s_{t'+lm(v')}.$$

Now, we define $t = t' + r = t' + (k_i - L_i)$ and we have

$$s_{k_i} = s_t, s_{k_i+1} = s_{t+1}, \ldots, s_{k_i+lm(v')-r-1} = s_{t+lm(v')-r-1},$$

$$\text{but } s_{k_i+lm(v')-r} \neq s_{t+lm(v')-r},$$

thus $Suff_{s\$}(k_i)$ and $Suff_{s\$}(t)$ have in common a longest prefix P of length $lm(v') - r = lm(v') - (k_i - L_i)$. Since B_{i-1} is deterministic by (A3), there is a unique state v where the reading of P ends. Since $lm(v') - (k_i - L_i) > lm(x_i)$, by (4), v is situated on $\beta_{k_i}[u_i \ldots leaf(k_i)]$. Moreover, we have $lm(v) = lm(v') - (k_i - L_i)$; otherwise, $lm(v) < lm(v') - (k_i - L_i)$ and $min(leaf(k_i))$ would be shorter than $Suff_{s\$}(k_i)$.

In addition,

$$\begin{array}{rcll}
|\overline{\beta_{k_i}}[u_i \ldots v]| & = & |\overline{\beta_{k_i}}[u_i \ldots leaf(k_i)]| - |\overline{\beta_{k_i}}[v \ldots leaf(k_i)]| & = \\
& = & |\overline{\beta_0}[u'_i \ldots leaf(L_i)]| - (m + 2 - k_i - lm(v)) & = \\
& = & |\overline{\beta_0}[u'_i \ldots leaf(L_i)]| - (m + 2 - L_i - lm(v')) & = \\
& = & |\overline{\beta_0}[u'_i \ldots leaf(L_i)]| - |\overline{\beta_0}[v' \ldots leaf(L_i)]| & = \\
& = & |\overline{\beta_0}[u'_i \ldots v']| &
\end{array} \tag{5}$$

The following equality also holds

$$\begin{array}{rcl}
|\overline{\beta_t}[v \ldots leaf(t)]| & = & m + 2 - t - lm(v) = m + 2 - t' - (k_i - L_i) - lm(v') + (k_i - L_i) = \\
& = & m + 2 - t' - lm(v') = |\overline{\beta_{t'}}[v' \ldots leaf(t')]|
\end{array} \tag{6}$$

Using (5), (6) and the remark (also based on (5) and (6)) that $\overline{\beta_t}[u_i \ldots leaf(t)]$, $\overline{\beta_{t'}}[u'_i \ldots leaf(t')]$ are suffixes of $s\$$ of the same length (and are therefore identical), we deduce that $\overline{\beta_{k_i}}[u_i \ldots v] = \overline{\beta_0}[u'_i \ldots v']$ and $\overline{\beta_t}[v \ldots leaf(t)] = \overline{\beta_{t'}}[v' \ldots leaf(t')]$.

To finish the proof of (A4a), notice that if t' is a weak suffix number of B_{i-1}, then so is t. Indeed, if $w' \neq t'$ is the suffix number of a branch which contains the parent of t', then $Suff_{s\$}(t)$ and $Suff_{s\$}(w' + (k_i - L_i))$ have a common prefix of length $|Suff_{s\$}(t)| - 1$, thus the branch accepting $Suff_{s\$}(w' + (k_i - L_i))$ contains the parent of t. The suffix number t is therefore weak. In the case where t' is a strong suffix number, t can be either a strong or a weak suffix number, as shown by Figure 1: in B_0, $t' = 5$ is strong and $t = 6$ is strong; in B_1, $t' = 6$ is strong while $t = 7$ is weak.

We will now prove statement (A4b). As mentioned above and because of (A2), in B_{i-1}:

$$Suff_{s\$}(t') = min(v')\overline{\beta_{t'}}[v' \ldots leaf(t')] = min(u'_i)\overline{\beta_0}[u'_i \ldots v']\overline{\beta_{t'}}[v' \ldots t'].$$

Once the contraction is performed, the path P_0 which allowed to read $min(u'_i)$ in B_{i-1} still exists in B_i, but its length is greater than $min(u_i)$ (since $lm(u_i) < lm(u'_i)$ by (A2)). Thus, in B_i the path $P = P_0\beta_0[u_i \ldots v]\beta_t[v \ldots t]$ allows to recognize $Suff_{s\$}(t')$, using affirmation (A4a).

Proof of (A5). This is easily proved by induction on i. When $i = 0$, $B_i = ST(s)$ and the affirmation is obviously true.

If we assume the affirmation is true in B_{i-1}, then by (A4b) the suffixes which were accepted in each removed leaf $leaf(t')$ of B_{i-1} will be accepted by B_i in $leaf(t' + (k_i - L_i))$; this is true since all the paths allowing B_{i-1} to accept such suffixes are replaced with equivalent paths (having different states but spelling the same word) in B_i.

Similarly, the suffixes which were accepted by B_{i-1} in $leaf(k_i)$ and $leaf(L_i)$ will be accepted by B_i in $leaf(L_{i+1})$.

Finally, consider the suffixes accepted in some $leaf(t')$ such that t' is free both in B_{i-1} and in B_i. They will be accepted in the same leaf $leaf(t')$ of B_i.

Theorem 1 is proved. ∎

Now, we assume that the $m+1$ states of $B(s)$, as well as the $m+1$ states of $SO(s)$ are denoted $0, 1, \ldots, m$ such that the internal transitions are $(i, s_{i+1}, i+1)$ in both cases $(i = 0, \ldots, m-1)$. As in [1], denote by $poccur(w, s)$ the last position of the first occurrence of the word w in s.

For a state x, denote $min_{SO}(x)$ and $min_B(x)$ the minimum word whose reading by the automaton $SO(s)$ and $B(s)$ respectively ends in x. Then, if i is a state of $SO(s)$, we have $i = poccur(min_{SO}(i), s)$ [1].

We can now affirm that:

Theorem 2 *The automaton $B(s)$ built by the Algorithm* **ST-to-SO** *for a given sequence s is the suffix oracle of s.*

Proof. We first need two claims. The first one is implicit in [1] and can be easily deduced using the property $x = poccur(min_{SO}(x), s)$.

Claim 1 [1] *In the factor/suffix oracle, if $x < u$ are two states and $\gamma \in \Sigma$, then there exists a transition (x, γ, u) if and only if $u = poccur(min_{SO}(x)\gamma, s)$.*

Claim 2 *In the automaton $B(s)$, if $x < u$ are two states and $\gamma \in \Sigma$, then there exists a transition (x, γ, u) if and only if $u = poccur(min_B(x)\gamma, s)$.*

Proof of Claim 2. Recall that, according to Algorithm **ST-to-SO**, the word read along β_0 is $s\$$ in all the automata B_i.

"⇒": Let $(i, L_i, k_i, x_i, \gamma_i, u_i, u'_i)$ be the step in Algorithm **ST-to-SO** which adds the transition (x, γ, u). This step transforms B_{i-1} into B_i. Then the current automaton is B_{i-1} and $x = x_i, u = u_i$ and $\gamma = \gamma_i$ (see Figure 2). By (A2) the minimum word read in x_i is unique. By (A1b) this word will not be changed in the next steps (which contract states deeper than x_i), so that this word is necessarily $min_B(x_i)$. Moreover, since B_{i-1} is deterministic by (A3), there is exactly one transition in B_{i-1} outgoing from x_i and labeled γ_i: the one on β_{k_i}.

By (A2) we have that

$$\begin{aligned} Suff_{s\$}(k_i) = \quad & min(leaf(k_i)) = min_B(x_i)\overline{\beta_{k_i}}[x_i \ldots leaf(k_i)] = \\ = \quad & min_B(x_i)\gamma_i\overline{\beta_{k_i}}[u_i \ldots leaf(k_i)] = min_B(x_i)\gamma_i\overline{\beta_0}[u'_i \ldots leaf(L_i)]. \end{aligned}$$

Consequently, since $Suff_{s\$}(k_i)$ is a suffix of $Suff_{s\$}(L_i)$ (and thus of $s\$$, which is read along β_0), we deduce that one can read on β_0 just before u'_i the word $min_B(x_i)\gamma_i$, and also that this occurrence of $min_B(x_i)\gamma_i$ in $s\$$ starts in position k_i. One has to show that this is the first occurrence of $min_B(x_i)\gamma_i$ in $s\$$ (and thus in s).

By contradiction, if this is not the case, then another occurrence of $min_B(x_i)\gamma_i$ exists in $s\$$ in position $k' < k_i$. By (A3) we have that B_{i-1} is deterministic, thus the path U which accepts $Suff_{s\$}(k')$ contains u_i and uses the path P which spells $min_B(x_i)$. Then, by (A5), $Suff_{s\$}(k')$ is accepted in a leaf $leaf(j)$ (with $j \geq k_i$) in the subtree of u_i. But then $Suff_{s\$}(k') = Suff_{s\$}(j)$. Indeed, (by (A2), $Suff_{s\$}(j)$ also uses P and the unique path going from u_i to $Suff_{s\$}(j)$. Obviously, this is not possible since $k' < k_i \leq j$ and thus $|Suff_{s\$}(k')| > |Suff_{s\$}(j)|$.

"⇐": Let $u = poccur(min_B(x)\gamma, s)$. By (A5), all the suffixes of s are accepted by $B(s)$, and thus the suffix of s starting in position $u - |min_B(x)\gamma| + 1$, which begins by $min_B(x)\gamma$, is accepted too. Since $B(s)$

is deterministic, there must exist a transition outgoing from x (where $min_B(x)$ is read) and labeled γ. If $u = x + 1$, the transition is internal and we are done. If $u > x + 1$, the transition must be external. According to the affirmation "⇒" proved above, the end state of this transition is necessarily u. ∎

To finish the proof of Theorem 2, denote by $SO[0 \dots x]$ ($B[0 \dots x]$ respectively) the partial automaton obtained from $SO(s)$ (from $B(s)$ respectively) by removing all states y with $y > x$ and all transitions incident to such a state.

We show by induction on the state $x \geq 0$ that:

(i_x) $SO[0 \dots x]$ is identical to $B[0 \dots x]$;

(ii_x) the transitions outgoing from the state x in $SO(s)$ and in $B(s)$ are exactly the same.

When $x = 0$, (i_x) is obviously true and $min_{SO}(x) = min_B(x) = \lambda$ (the empty word). By Claims 1 and 2, both in $SO(s)$ and $B(s)$, for each $\gamma \in \Sigma$ which appears in s there is a transition labeled γ from x to the minimum state u such that $s_u = \gamma$. All these transitions are external, except for the case $\gamma = s_1$. Affirmation (ii_x) is then proved.

Assume the affirmations $(i_{x'})$ and $(ii_{x'})$ are true for $x' = 0, 1, 2, \dots, x-1$ and let us prove them for $x \geq 1$. By $(ii_{x'})$, all the transitions outgoing from states $x' < x$ and incoming to states $u' \leq x$ are the same, so that (i_x) holds. Consequently, $min_{SO}(x) = min_B(x)$ and thus, by Claims 1 and 2, affirmation (ii_x) holds.

The states and the transitions of $B(s)$ and $SO(s)$ are thus identical. To conclude that the two automata are identical, one has to show that the terminal states are the same. Denote B_h the last automaton built in the *while* loop of Algorithm **ST-to-SO**. Then $B(s)$ is obtained from B_h by removing some leaves and marking their parents as terminal. By (A5), in B_h every suffix of $s\$$ is accepted in a leaf of B_h, which is either weak (since the *while* loop is finished) or is $leaf(L_{h+1})$. Thus, according to the algorithm, every suffix of s is accepted by $B(s)$ in a terminal state. Conversely, a state of $B(s)$ is terminal only if it is the parent of a weak leaf of B_h or of $leaf(L_{h+1})$; each of these leafs ends the recognition of at least one suffix of $s\$$. Thus the terminal states of $B(s)$ are the same as those of $SO(s)$. This finishes the proof of Theorem 2. ∎

Several corollaries can be easily deduced from this main result. The first one concerns the arrival order of the external arcs incoming the same state u of $B(s)$. According to (A1b), these external arcs are added in consecutive steps of Algorithm **ST-to-SO**. The result below shows that each new added arc is *longer* that the preceding one, that is, its initial state is less deep than the initial state of the preceding arc (while the end state is the same, u).

Corollary 1 *In Algorithm* **ST-to-SO***, if there exist integers $i > l \geq 1$ such that $u_{i-l}, u_{i-l+1}, \dots, u_{i-1}, u_i$ are contracted in the same state u_i of $B(s)$, then the states $x_i, x_{i-1}, \dots, x_{i-l+1}, x_{i-l}$ are distinct and occur in this order on the main branch.*

Proof. Notice first that, since $u_{i-l}, u_{i-l+1}, \dots, u_{i-1}, u_i$ are contracted in the same state and since the automaton is homogeneous at each step by (A3), we must have $\gamma_{i-l} = \gamma_{i-l+1} = \dots = \gamma_{i-1} = \gamma_i$. We denote all these values by γ.

By contradiction, assume that Corollary 1 is not true. Then, there exists j, $1 \leq j \leq l$, such that x_{i-j} is less deep than x_{i-j+1}. In B_{i-j+1} (and therefore in $B(s)$) we obtain the transitions in Figure 4a). In B_{i-j-1} (see Figure 4b), one has

$$\begin{aligned} Suff_{s\$}(k_{i-j}) &= min_B(x_{i-j})\gamma\overline{\beta_{k_{i-j}}}[u_{i-j} \dots leaf(k_{i-j})] = \\ &= min_B(x_{i-j})\gamma\overline{\beta_{k_{i-j+1}}}[u_{i-j+1} \dots leaf(k_{i-j+1})] \end{aligned} \tag{7}$$

since $\overline{\beta_{k_{i-j}}}[u_{i-j} \dots leaf(k_{i-j})]$ and $\overline{\beta_{k_{i-j+1}}}[u_{i-j+1} \dots leaf(k_{i-j+1})]$ represent suffixes of s of the same length, thus they spell the same word. Moreover,

$$Suff_{s\$}(k_{i-j+1}) = min_B(x_{i-j+1})\gamma\overline{\beta_{k_{i-j+1}}}[u_{i-j+1} \dots leaf(k_{i-j+1})]. \tag{8}$$

Now, because of $k_{i-j} = L_{i-j+1} < k_{i-j+1}$, we have that $|Suff_{s\$}(k_{i-j})| > |Suff_{s\$}(k_{i-j+1})|$, therefore $Suff_{s\$}(k_{i-j+1})$ is a suffix of $Suff_{s\$}(k_{i-j})$. Consequently, using (7) and (8), $min_B(x_{i-j+1})$ is a suffix of

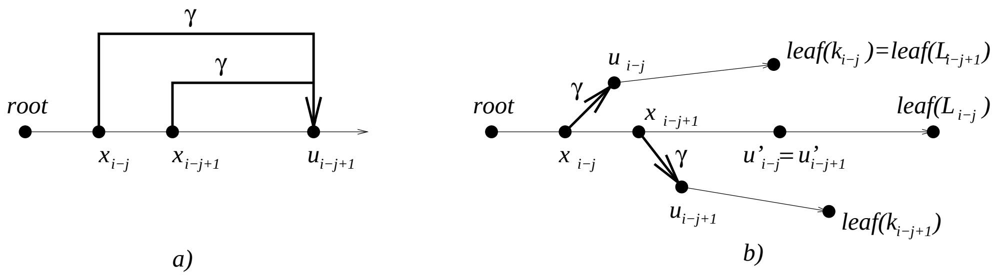

Figure 4: *Automata $B(s)$ and B_{i-j-1} in the proof of Corollary 1.*

$min_B(x_{i-j})$. Then we can use the following statement (recall that the factor oracle is the suffix oracle where all the states are terminal):

Claim 3 ([1]) *Let w be a factor of s.Then every suffix of w is recognized by the factor oracle of s in a state $l < poccur(w)$.*

With $w = min_B(x_{i-j})$ and since $min_B(x_{i-j+1})$ is a suffix of w, we deduce that state x_{i-j+1}, which ends the spelling of $min_B(x_{i-j+1})$ in $B(s)$, satisfies $x_{i-j+1} < poccur(min_B(x_{i-j}))$. But $poccur(min_B(x_{i-j})) = x_{i-j}$, so that $x_{i-j+1} < x_{i-j}$. But this contradicts the initial assumption that $x_{i-j+1} > x_{i-j}$. ■

The two corollaries below allow to estimate the number of external arcs. The second one gives a new proof of an already known result.

Corollary 2 *The number of external transitions in $SO(s)$ is upper bounded by the number of maximal suffixes of s.*

Proof. The suffixes of s which are not maximal give weak leaves in $ST(s)$. None of the weak leaves yields an external transition in $B(s)$ during the algorithm. Each other leaf (but the leaf 1) is either bent (and thus it gives an external transition) or is removed during the contractions.■

Corollary 3 [1] *The total number of transitions $T_{SO}(s)$ of $SO(s)$ satisfies $|s| \leq T_{SO}(s) \leq 2|s| - 1$.*

Proof. This can be easily proved using Corollary 2 and the remark that s can have at most $|s| - 1$ maximal suffixes■

5 Complexity results

In this section we assume that $ST(s)$ is implemented in its compact version, and that the construction of the automata B_i preserves the compactness of the branches.

Theorem 3 *Algorithm* **ST-to-SO** *runs in $O(|s|)$.*

Proof. The number of operations the algorithm performs before and after the *while* loop is clearly linear. In the *while* loop (which is executed at most $|s| - 1$ times), only two tasks need more than a constant time to be executed: finding x_i and finding u'_i (notice that β_0 does not need to be explicitly computed). We show that globally (over all the executions of the *while* loop), each of these tasks needs linear time. To do this, we will use the following result, where B_h denotes the last automaton built in the *while* loop of Algorithm **ST-to-SO**:

Claim 4 *With $minb(k_i) = |\beta_{k_i}[h(k_i) \ldots leaf(k_i)]|$ ($i = 1, \ldots, h$), we have that $\Sigma_{i=1}^{h} minb(k_i)$ is in $O(|s|)$.*

Proof of Claim 4. Consider B_{i-1} and recall that, by (A3a), $x_i = h(k_i)$ and $minb(k_i)$ is the same in B_{i-1} and in $B_0 = ST(s)$. Then $minb(k_i)$ is the length of the path $\beta_{k_i}[h(k_i) \ldots leaf(k_i)]$ in $ST(s)$. If we consider this path backwards, we obtain a walk in $ST(s)$ starting with $leaf(k_i)$ and advancing on β_{k_i} until a branch with smaller number is encountered. Now, $\Sigma_{i=1}^{h} minb(k_i)$ is the total length of the backward walks along *all* the paths $\beta_{k_i}[h(k_i) \ldots leaf(k_i)]$ $(i = 1, 2, \ldots, h)$ in $ST(s)$. It is easy to see that these walks are equivalent (up to the traversal of the branch β_1) to a depth first traversal of $ST(s)$ where the children of each node are considered in increasing order. The global time needed to perform this is thus linear, if we consider that the implementation uses the compact suffix tree∎

Now we consider the two tasks identified above. To perform them in linear time, one needs a pre-treatment of $ST(s)$ in order to compute $minb(j)$ $(2 \leq j \leq h)$, and the length $lm(v)$ of $min(v)$ for every state v of $ST(s)$. According to (A1) and (A3a) respectively, these values remain constant in the other automata B_i built by the algorithm. The values $minb(j)$ $(2 \leq j \leq h)$ and $lm(v)$ (v is a state of $ST(s)$) are easily computed using a depth-first traversal of $ST(s)$.

Finding x_i. By (A3a), a backwards traversal of β_{k_i} starting with $leaf(k_i)$ and using $minb(k_i)$ transitions will stop on $x_i = h(k_i)$. Using Claim 4, the backwards traversals to find *all* the values x_i need linear time.

Finding u'_i. Notice that $\overline{\beta_{k_i}}[u_i \ldots leaf(k_i)]$ and $\overline{\beta_0}[u'_i \ldots leaf(L_i)]$ are both suffixes of $s\$$. Then it is sufficient to compute the length $l = lm(leaf(k_i)) - lm(u_i)$ of the word spelled by $\beta_{k_i}[u_i \ldots leaf(k_i)]$ and to identify the state u'_i using a backwards traversal of β_0 until the needed length is reached. This traversal takes at most $minb(k_i) = |\beta_{k_i}[x_i \ldots leaf(k_i)]|$ steps since by (A4a) every state v' on $\beta_0[u'_i \ldots leaf(L_i)]$ has a *copy* on $\beta_{k_i}[u_i \ldots leaf(k_i)]$, therefore $|\beta_{k_i}[u_i \ldots leaf(k_i)]| > |\beta_0[u'_i \ldots leaf(L_i)]|$. Claim 4 finishes the proof.∎

6 Perspectives

The algorithm we proposed transforms a very well known structure (the suffix tree) into a structure which was introduced recently and is about to be studied (the factor/suffix oracle). In this context, the simplicity of the contraction operation and the successive steps in the algorithm should allow to deduce new results on oracles starting with already known result on suffix trees. Typically, statistical results on the number of words validated by the oracles could possibly be obtained in this way.

References

[1] C. Allauzen, M. Crochemore, M. Raffinot - Efficient experimental string matching by weak factor recognition, *In* Proceedings of CPM'2001, LNCS 2089 (2001), 51-72.

[2] L. Cleophas, G. Zwaan, B. W. Watson - Constructing Factor Oracles, *In* Proceedings of the Prague Stringology Conference 2003 (PSC '03), Czech Technical University, Prague, 37-50.

[3] M. Crochemore - Reducing space for index implementation, *Theoretical Computer Science*, 292 (2003), 185-197.

[4] R. Kato - A new full-text search algorithm using factor oracle as index, TR-C185, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Japan.

[5] R. Kato - Finding mximal repeats with factor oracles, TR-C190, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Japan.

[6] T. Lecroq, A. Lfebvre - Computing repeated factors with a factor oracle, In *Proceedings of the 11th Australasian Workshop On Combinatorial Algorithms*, L. Brankovic and J. Ryan eds. (2000), 145-158.

[7] T. Lecroq, A. Lfebvre - Compror: on-line lossless data compression with a factor oracle, *Information Processing Letters* 83 (2002), 1-6.

[8] A. Mancheron, C. Moan - Combinatorial characterization of the language recognized by factor and suffix oracles, to appear in *International Journal of Foundations of Computer Science*.

Converting Suffix Trees into Factor/Suffix Oracles

Irena Rusu

Abstract

The factor/suffix oracle is an automaton introduced by Allauzen, Crochemore and Raffinot. It is built for a given sequence s on an alphabet Σ, and it *weakly recognizes* all the factors (the suffixes, respectively) of s : that is, it certainly recognizes all the factors (suffixes, respectively) of s, but possibly recognizes words that are not factors (not suffixes, respectively) of s. However, it can still be suitably used to solve pattern matching and compression problems. The main advantage of the factor/suffix oracle with respect to other indexes is its size: it has a minimum number of states and a very small (although not minimum) number of transitions.

In this paper, we show that the factor/suffix oracle can be obtained from another indexing structure, the suffix tree, by a linear algorithm. This improves the quadratic complexity previously known for the same task.

General Terms: Algorithms

Additional Key Words and Phrases: indexing structure, factor recognition, suffix recognition

LINA, Université de Nantes
2, rue de la Houssinière
B.P. 92208 — F-44322 NANTES CEDEX 3